Una Aproximación Dirigida por Modelos para el Desarrollo de Bases de Datos Objeto-Relacionales

Verónica A. Bollati, Belén Vela, Juan M. Vara, Esperanza Marcos

Kybele Research Group
Rey Juan Carlos University
Madrid (Spain)
{veronica.bollati, belen.vela, juanmanuel.vara, esperanza.marcos}@urjc.es

Resumen. Este artículo propone una aproximación dirigida por modelos para el desarrollo de Bases de Datos (BD) Objetos-Relacionales (OR). El punto de partida del proceso es un modelo conceptual de datos representado mediante un diagrama de clases UML. Tomando como entrada dicho modelo y aplicando un conjunto de reglas de transformación (modelo a modelo) se obtiene un nuevo modelo que representa el esquema de la BDOR. Finalmente, una transformación modelo a texto genera el *script* SQL que implementa el esquema de la BDOR. La propuesta se completa con su implementación en la herramienta M2DAT (*MDA MIDAS Tool*), dándole soporte tecnológico a la propuesta y permitiendo así automatizar el proceso de desarrollo de BDOR.

Palabras Claves: Desarrollo de Software Dirigido por Modelos, Bases de Datos Objeto-Relacionales, Transformaciones de Modelos, Generación de Código.

1 Introducción

A pesar del impacto de las Bases de Datos (BD) Relacionales en las últimas décadas, éstas tienen algunas limitaciones a la hora de dar soporte a la persistencia de datos requerida por las aplicaciones actuales. Debido a la mejora del hardware han surgido aplicaciones más sofisticadas, las cuales se caracterizan por poseer objetos y relaciones complejas. Para representar cada objeto y sus relaciones en el modelo relacional se debe descomponer el objeto en un determinado número de tuplas. De esta manera, a la hora de recuperar un objeto es necesario realizar un conjunto considerable de *joins*, y cuando los objetos que se desean recuperar son demasiados complejos, el rendimiento se reduce considerablemente [3]. Para resolver este problema nace una nueva generación de BD: las BD Orientadas a Objetos (OO), que incluye a las BD Objeto-Relacionales (OR) [21].

Esta tecnología, basada en estándares [9], permite almacenar y recuperar datos complejos, ya que soporta el uso de los tipos definidos por el usuario, tipos colección, tablas tipadas, generalizaciones, datos multimedia, etc. Actualmente las BDOR se utilizan en el ámbito industrial y en el de investigación, lo que ha hecho que una gran cantidad de productos comerciales [8,15,19] las soporten.

Sin embargo, no es suficiente con tener una buena tecnología, es necesario establecer metodologías que orienten a los diseñadores de BDOR en la tarea de desarrollo, tal como se ha hecho tradicionalmente con las BD relacionales [6]. Estas metodologías deben incorporar el modelo OR, teniendo en cuenta los antiguos y los nuevos problemas, como son: la elección de la tecnología adecuada, la migración y la independencia de plataforma, el mantenimiento, etc.

En este trabajo se aplican las ideas del Desarrollo de Software Dirigido por Modelos (DSDM) [20] al desarrollo de BDOR. La base del DSDM es el uso de modelos en distintos niveles de abstracción para representar un Sistema de Información (SI) en las distintas fases del proceso de desarrollo y la definición de reglas de transformación entre dichos modelos.

En el proceso de desarrollo que se propone, se parte de un Modelo Independiente de Plataforma (*Platform Independent Model*, PIM) al que se le aplican un conjunto de reglas de transformación de modelo a modelo (*Model to Model*, M2M) formalizadas e implementadas, dando lugar al Modelo

Específico de Plataforma (*Platform Specific Model*, PSM) que será el esquema de la BDOR. Finalmente, aplicando una transformación de modelo a texto (*Model to Text*, M2T) se obtiene el *script* con el código SQL que implementa el esquema de la BDOR en el producto Oracle10g. Este proceso para el DSDM de BDOR ha sido implementado dentro de una herramienta que da soporte a la metodología dirigida por modelos de MIDAS, como se explicará en las siguientes secciones.

El resto del trabajo se estructura de la siguiente manera: en primer lugar, en la sección 2 se presenta de forma resumida la propuesta para el proceso de desarrollo de BDOR incluyendo el metamodelo OR y las transformaciones de PIM a PSM. En la sección 3 se presenta un caso de estudio que muestra la implementación de la propuesta en la herramienta desarrollada. Finalmente, en la sección 4 se plantean las principales conclusiones y se presentan los futuros trabajos.

2 Desarrollo de BD Objeto-Relacionales en MIDAS

Nuestra propuesta está enmarcada en MIDAS [11], una metodología dirigida por modelos para el desarrollo de SI. Como se muestra en la Fig. 1, MIDAS propone el modelado de SI de acuerdo a dos dimensiones ortogonales. Por un lado, de acuerdo con los principios de MDA (*Model Driven Architecture*) [18], MIDAS define diferentes modelos para la especificación del sistema en función del nivel de abstracción: modelos independientes de computación (*Computation Independent Models*, CIM), PIM y PSM. Por otro lado, se consideran tres aspectos básicos para el desarrollo de SI: hipertexto, contenido y comportamiento. La definición de aspectos incrementa la escalabilidad en el desarrollo de SI, ya que se pueden incluir nuevas características o funcionalidades añadiendo nuevos aspectos a la arquitectura del sistema.

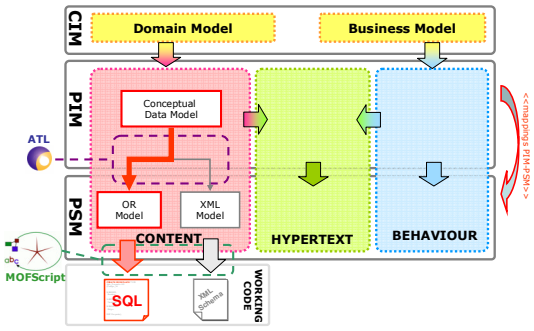

Fig. 1. Arquitectura simplificada de MIDAS: Desarrollo de BDOR

Este trabajo se centra en el aspecto de *contenido*, que se corresponde con el concepto tradicional de BD, y cuyos modelos en los niveles PIM y PSM se muestran en la Fig. 1. A nivel PIM se define el modelo conceptual de datos representado mediante un diagrama de clases UML. A nivel PSM se definen dos modelos, el modelo OR y el modelo de esquemas XML, dependiendo de la tecnología seleccionada para la implementación de la BD. Para el caso de la tecnología OR se consideran dos PSMs diferentes: el primero basado en el estándar SQL:2003 [9] y el segundo para un producto específico, Oracle10g [19]. En este trabajo se presenta el proceso de desarrollo completo para BDOR; el proceso de desarrollo para BD XML se puede consultar en [5, 26]. Se han definido los dos metamodelos para el modelado de BDOR, uno para el estándar SQL:2003 y otro para el producto Oracle10g, este último se mostrará en la siguiente subsección. En este artículo no se ha incluido el metamodelo para el estándar SQL:2003, ya que éste es muy similar al del producto Oracle10g. Además a partir del PSM definido para el producto Oracle10g se puede generar código que será directamente implementable en el producto.

Para completar el proceso de desarrollo se han definido las transformaciones de PIM a PSM (M2M) y las transformaciones que permiten obtener el código SQL para la BDOR (M2T). Así, a partir de la definición del PIM y aplicando las transformaciones definidas, se puede obtener el código SQL de la BD para el producto Oracle10g.

Todo el proceso para el DSDM de BDOR se ha integrado en la herramienta M2DAT (*MIDAS MDA Tool*), que da soporte completo a la metodología MIDAS para la generación semi-automática de SI. La naturaleza modular de MIDAS facilita el desarrollo modular de M2DAT. Por lo tanto, para el desarrollo de esta herramienta, se están abordando cada una de las propuestas de MIDAS como módulos separados, los cuales posteriormente se integran a través de un conjunto de transformaciones. En cuanto a las soluciones técnicas utilizadas en la herramienta M2DAT para soportar este proceso de desarrollo, todas ellas se enmarcan dentro del *Eclipse Modeling Project* (EMP), un proyecto de Eclipse que agrupa las iniciativas dirigidas a proporcionar herramientas de soporte para el DSDM, todas ellas construidas sobre la base común del *Eclipse Modeling Framework* (EMF). Así, dentro del EMP se pueden encontrar herramientas para la construcción de editores como el *Graphical Modeling Framework* (GMF), motores de transformación, como ATL o VIATRA, generadores de código como MOFScript o Xpand, o implementaciones de estándares, como UML2 u OCL. En nuestro caso, se ha optado por utilizar ATL [10] para implementar las transformaciones M2M y MOFScript [16] para la implementación de las transformaciones M2T. Más adelante se justificarán estas decisiones. Actualmente la herramienta continúa en la fase desarrollo y en este trabajo se presenta una parte de su funcionalidad (la correspondiente al DSDM de BDOR), aunque algunas de sus primeras funcionalidades ya han sido descritas en [23, 24, 25].

A continuación, se presentan el metamodelo definido para el modelado de BDOR (para el producto Oracle 10g) y se resumen las reglas de transformación entre los niveles PIM y PSM.

2.1 Modelado de BDOR

El modelado de BDOR implica la definición del metamodelo OR, para lo que en este trabajo se ha optado por definir un Lenguaje Específico de Dominio (*Domain Specific Languages*, DSL) [13].

Antes de tomar esta decisión y con el objetivo de seguir los principios impuestos por el DSDM, se estudiaron las dos aproximaciones existentes: utilización de perfiles UML o definición de nuevos lenguajes basados en MOF. Inicialmente se definieron sendos perfiles UML para el modelado de BDOR [12,23] y esquemas XML [27]. Pero posteriormente, a la hora de desarrollar el soporte tecnológico para el proceso completo descrito en la sección anterior, se optó por el uso de DSLs.

Esta decisión se debe fundamentalmente a que las facilidades proporcionadas por el EMP y otros *frameworks* para el trabajo con DSLs, como el *Generic Modeling Environment* (GME) o las DSL *Tools,* han potenciado la aparición de propuestas basadas en MOF [4,13]. De hecho, metodologías contrastadas, que inicialmente se basaban en el uso de perfiles UML y herramientas *ad-hoc* (como WebML y su herramienta WebRatio[1]) están siendo actualizadas y/o redefinidas para utilizar lenguajes basados en MOF e integrarse en los *frameworks* mencionados. En teoría, la apuesta por lenguajes basados en MOF resulta en una pérdida de interoperabilidad, pues obliga a desarrollar nuevas transformaciones de modelos cuando se quiere cambiar el DSL utilizado para modelar el SI. En la práctica, el mismo problema subyace cuando se usan perfiles UML, pues a día de hoy la interoperabilidad prometida por XMI sigue sin haberse materializado.

Por lo tanto, teniendo en cuenta la tecnología existente, parece más apropiado expresar los conceptos relacionados con el DSDM para BDOR mediante dos metamodelos basados en MOF, uno para el estándar y otro para el producto Oracle 10g. Como ya se ha dicho antes, en este trabajo sólo se mostrará el el metamodelo del producto Oracle10g (ver Fig. 2). Las principales diferencias

con el metamodelo propuesto para el estándar son: Oracle no brinda soporte para el tipo *Row* ni para la herencia de tablas, pero soporta el tipo *Nested Table* que representa aproximadamente el mismo concepto que el tipo *Multiset* incluido en el estándar SQL:2003.

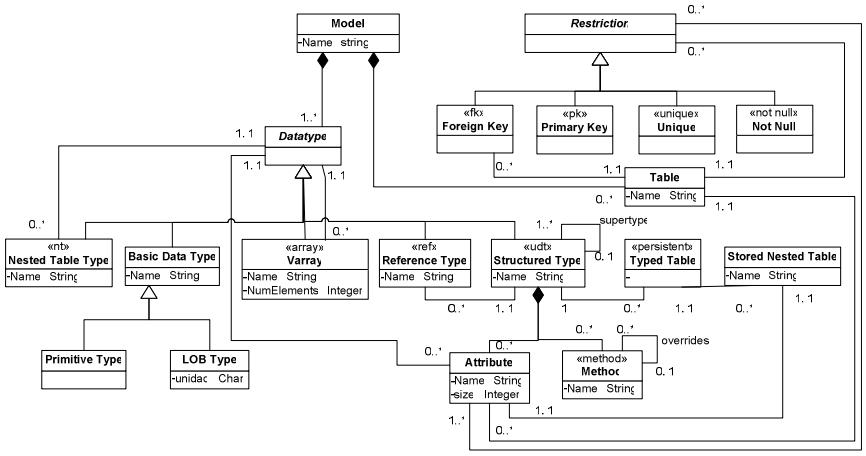

Fig. 2. Metamodelo OR para Oracle 10g's

2.2 Transformaciones de PIM a PSM para el desarrollo de BDOR

Dentro del proceso de desarrollo para BDOR se han definido un conjunto de reglas de transformación para pasar del modelo conceptual de datos (PIM) al modelo OR (PSM). En cuanto a la forma en la que deben definirse, en [18] se dice que "la descripción de las transformaciones puede hacerse utilizando lenguaje natural, algoritmos o modelos de transformaciones". De esta manera, en trabajos anteriores [23] se ha esbozado un enfoque para abordar el desarrollo de las transformaciones entre modelos en MIDAS:

- Primero, se definen las transformaciones entre modelos utilizando lenguaje natural.
- Luego, estas reglas de transformación son formalizadas usando gramática de grafos.
- Por último, las reglas resultantes son implementadas utilizando alguna de las propuestas de modelos de transformaciones existentes. Para este trabajo se ha seleccionado el lenguaje ATL, como se ha indicado en el apartado anterior.

Esta propuesta está orientada a dar solución a algunos problemas que se han detectado en el ámbito de las transformaciones de modelos: la brecha existente entre los desarrolladores y las diferentes propuestas de modelos de transformaciones a la hora de seleccionar cuál es la más conveniente. El objetivo de este trabajo es tratar de reducir esta brecha mediante la propuesta de un método sencillo para la definición de transformaciones. El hecho de utilizar grafos [2] para la formalización de las reglas de transformación antes de su implementación facilita la detección de errores e inconsistencias en las primeras fases de desarrollo y ayuda a aumentar la calidad de los modelos construidos, así como el código generado a partir de los mismos. De igual modo, la formalización de transformaciones simplifica significativamente el desarrollo de herramientas de apoyo a cualquier enfoque dirigido por modelos.

Como ya se ha dicho, para la implementación de las reglas de transformación se ha seleccionado el lenguaje ATL [10], un lenguaje de transformación de modelos desarrollado por el grupo ATLAS. ATL se basa principalmente en el estándar OCL [17] y soporta tanto el enfoque declarativo como el imperativo, aunque se recomienda el uso del enfoque declarativo del mismo. Se ha seleccionado ATL, porque actualmente es considerado el estándar de facto para las transformaciones de modelos. La usabilidad de *Query/View/Transformations* (QVT) [14], propuesto por el grupo OMG, es muy compleja, lo que se constata por el hecho de que no existe ninguna implementación completa del

mismo. Por otro lado, ATL provee un conjunto de herramientas para el desarrollo de transformaciones de modelos. Además, su gran comunidad de usuarios proporciona información continua para la mejora del motor de transformación de modelos. Para implementar las transformaciones en ATL se definen un conjunto de reglas, cada regla identifica el elemento de entrada (*source pattern*) y el elemento de salida (*target pattern*), ambos a nivel de metamodelo. Cuando se ejecuta la transformación ATL, el motor establece las relaciones entre los elementos de entrada y el modelo de entrada. Luego, para cada relación existente se instancia un elemento de salida en el modelo de salida. En la sección 3 se mostrará la implementación de algunas de las reglas de transformación utilizando ATL para el caso de estudio seleccionado.

En la tabla 1 se resumen las reglas de transformación de PIM a PSM para la propuesta de DSDM para BDOR. Estas reglas han sufrido un proceso continuo de refinamiento, de hecho la primera versión de las mismas fue pensada para el estándar SQL:1999 y para la versión 8i de Oracle [12].

Tras su definición en lenguaje natural, el siguiente paso ha sido la formalización de las reglas de transformación definidas, utilizado la propuesta de transformaciones de grafos [2]. La formalización completa de las reglas de transformación entre PIM y PSM para el desarrollo de modelos OR usando transformación de grafos se puede consultar en [23].

Tabla 1. Transformaciones OR de PIM a PSM

PIM de Datos		**PSM de Datos Estándar (SQL:2003)**	**PSM de Datos para Producto (Oracle10*g*)**
Clase		Structured Type + Typed Table	Object Type + Object Table
Atributos	Simple	Attribute (column)	Attribute (column)
	Multivaluado	Array/Multiset	Varray/Nested Table
	Compuesto	ROW/Structured Type (column)	Object Type (column)
	Calculado	Trigger/Method	Trigger/Method
Asociaciones	1:1	Ref/Ref	Ref/Ref
	1:M	Ref - Multiset/Array	Ref - Nested Table/Varray
	N: M	Multiset/Multiset - Array/Array	Nested Table/Nested Table - Varray/Varray
	Agregación	Multiset/Array	Nested Table/Varray of References
	Composición	Multiset/Array	Nested Table/Varray of Objects
	Generalización	Types/Typed Tables	Types/Typed Tables

3 Caso de Estudio

En esta sección se presenta la propuesta para el DSDM para BDOR a través del desarrollo de un caso de estudio: una BDOR para la gestión de la información de los proyectos de un estudio de arquitectura. Como ya se ha indicado, en primer lugar se define el modelo conceptual de datos (PIM) representado mediante un diagrama de clases UML (sección 3.1). Posteriormente, partiendo de este PIM y aplicando las reglas implementadas con ATL se genera el PSM para la BDOR (secciones 3.2 y 3.3). Por último, a partir de dicho PSM se obtiene el código SQL usando el *script* de MOFScript definido (sección 3.4).

En nuestro caso, una vez definido el PIM de datos, es la herramienta la que realiza de forma automática el proceso de transformación completo. Sin embargo, el diseñador puede refinar y/o modificar el modelo OR generado antes de comenzar con la etapa de generación de código, usando para ello el editor gráfico de modelos conforme al metamodelo mostrado en la sección 2.1, que incorpora la herramienta M2DAT.

3.1 Modelo Conceptual de Datos

Como primer paso del proceso de desarrollo de la BDOR para el caso de estudio elegido, se define el PIM, (ver Fig. 3): un Jefe de Proyecto (*Manager*) dirige un proyecto, cada proyecto (*Project*) está compuesto de un conjunto de planos (*Plan*) que a su vez poseen un conjunto de figuras (*Figure*). Estas figuras pueden ser polígonos (*Poligon*) que están formadas por líneas (*Line*).

Para realizar la representación gráfica de este modelo en la herramienta M2DAT se usa UML2 [22], una implementación del metamodelo de UML basada en EMF para la plataforma de Eclipse.

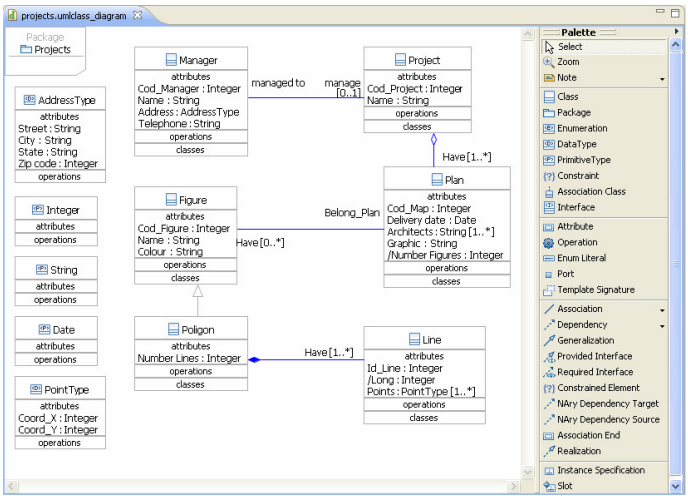

Fig. 3. Modelo Conceptual de Datos (PIM) para el Caso de Estudio

3.2 Transformaciones de PIM a PSM

A continuación, se muestran algunas de las reglas de transformación aplicadas al PIM para obtener el modelo OR para el producto Oracle10g. Para cada una de ellas, se presenta la formalización de cada regla con gramática de grafos y su correspondiente implementación en ATL.

Transformaciones de Clases y Propiedades. La parte izquierda de la Fig. 4 muestra las reglas de transformaciones de grafos para transformar clases persistentes (clases en el PIM) a elementos del esquema BDOR (clases en el PSM). Siempre que haya una clase UML persistente en el nivel PIM (①−③), se creará, a nivel PSM, un tipo estructurado (*structured type* u *object type*) y una tabla tipada (*typed table*) (①'). El tipo de la nueva tabla tipada deberá ser el tipo estructurado, por lo que la tabla tipada será una extensión del tipo estructurado. Por cada una de las propiedades (*property*) de la clase persistente se añadirá un atributo al tipo estructurado (②⇨②').

A la derecha de la Fig. 4 se muestran las reglas ATL *Class2UDT* y *Property2Attribute* que se corresponden con la transformación de grafos definida. La regla *Class2UDT* especifica que por cada clase encontrada en el modelo de origen (*UML!Class*) se debe crear un tipo estructurado y una tabla tipada en el modelo de destino (*modeloOR!StructuredType* y *modeloOR!TypedTable*), dándole además valor a las propiedades de los atributos del tipo estructurado como por ejemplo, la propiedad *Name*. Por otro parte, la regla ATL *Property2Attribute* transforma cada propiedad UML de la clase origen en un atributo del tipo estructurado al que pertenece. Para ello, la propiedad *structured* de cada atributo apunta a la clase origen dueña de la propiedad al comienzo de la transformación (lin. 30). Cuando el motor ATL evalúa esta expresión debe resolver la referencia al tipo estructurado que se corresponde con la clase origen en el modelo destino. De esta manera, se puede realizar la navegación en el modelo de destino cuando se necesita establecer las referencias entre elementos. El motor ATL se ocupa de esta tarea utilizando su mecanismo *resolve* [10]: cada

referencia a un elemento en el modelo de fuente se sustituye por una referencia al elemento transformado en el modelo destino.

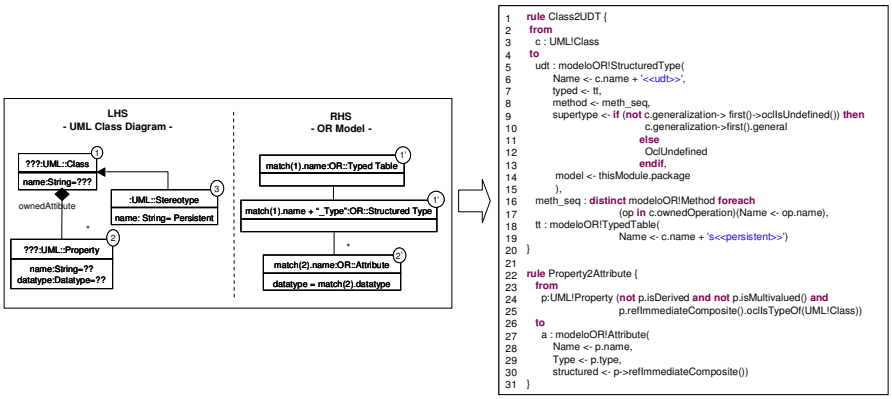

Fig. 4. Regla de Transformación de Clases Persistentes

En la Fig. 5 se muestra una instanciación de la regla. En la parte izquierda se muestra un extracto del modelo conceptual de datos: la clase UML *Manager* y sus propiedades. A la derecha de la figura se muestra, representado con el editor gráfico de M2DAT, el resultado de aplicar las reglas ATL: un tipo estructurado *Manager* con sus atributos y su correspondiente tabla tipada *Managers*.

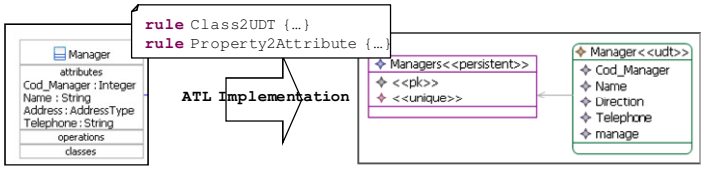

Fig. 5. Transformación de la clase *Manager*

Transformación de Atributos Multivaluados. Las reglas de transformación presentadas anteriormente sirven para transformar clases y sus atributos en general. Como se puede ver en la Tabla 1, existen además reglas específicas dependiendo del tipo de atributos del que se trate, es decir, se han definido reglas para transformar atributos multivaluados, compuestos y calculados o derivados. Por cuestiones de espacio, en este trabajo se presentará únicamente la regla de transformación para los atributos multivaluados.

Los atributos multivaluados del PIM se corresponden con atributos de tipo colección en el esquema BDOR. En Oracle10g existen dos constructores predefinidos para tratar colecciones de tipos: *VArray* y *Nested Table*. La principal diferencia entre éstos radica en que los *VArrays* tienen un tamaño máximo fijo y a la hora del almacenamiento se guardan en el mismo espacio que la tabla, mientras que las *Nested Tables* son de tamaño variable y se almacenan de forma independiente, asociándose a la tabla sobre la que está definida. En la Fig. 6 se muestra la regla de transformación de grafos para el caso en el que se transforme un atributo multivaluado a un atributo de tipo *Nested Table*: la clase UML (①) tiene una propiedad multivaluada (②), siendo el valor del atributo *isMultivalued true*. Por lo tanto, el tipo estructurado definido a partir de la clase persistente (①') tiene atributos de tipo *Nested Table* (②'−②'). El mismo caso se da si se usase el tipo *VArray*. En este punto, sería necesario permitir que el diseñador seleccione si quiere crear un *VArray* o una *Nested Table* en el momento de su transformación. Sin embargo, ATL no ofrece una manera natural de personalizar la transformación, permitiendo tomar decisiones de diseño. Así para este caso se ha elegido transformar los atributos multivaluados de forma genérica mediante un atributo de tipo *Nested Table*. Actualmente se está trabajando para resolver este problema usando, por ejemplo, modelos de anotación [7].

A la derecha de la Fig. 6 se muestran las reglas utilizadas para realizar la transformación de los atributos multivaluados. La regla *Property2MultiValuedAttribute* tiene un punto de entrada que

incluye una condición que sólo permite transformar los atributos multivaluados de las clases UML (lin. 3-4) y define que en el modelo de salida se deben crear dos elementos por cada propiedad encontrada en el modelo de origen: un atributo OR y un tipo *Nested Table* (lin. 11-25). Estos elementos se crean de forma paralela y la propiedad *type* del nuevo atributo es inicializada llamando a la regla *lazy generateNestedTableMultivalued* (lin. 8-9). Una regla *lazy* también es declarativa pero la diferencia es que debe ser invocada explícitamente. De esta manera, el tipo del atributo referenciará al tipo *Nested Table* creado.

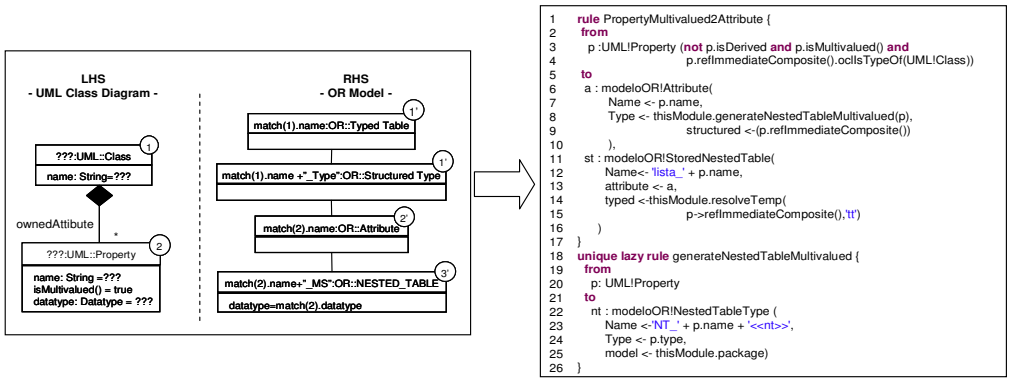

Fig. 6. Regla de Transformación de Atributos Multivaluados

En la Fig. 7 se muestra el resultado de la transformación del atributo multivaluado *Architects* de la clase *Plan*. Como se puede ver el tipo del atributo *Architects* del tipo estructurado Plan es del tipo *Nested Table NT_Architects*.

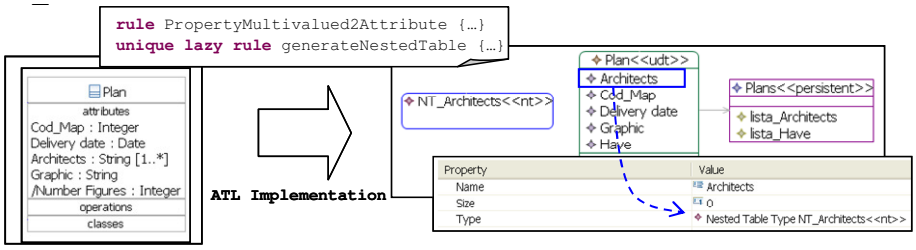

Fig. 7. Transformación de la propiedad multivaluada *Architects*

Transformación de Asociaciones. Teniendo en cuenta la multiplicidad de las clases involucradas en la asociación, se proponen diferentes reglas (ver Tabla 1). Por cuestiones de simplicidad, en este trabajo sólo se contemplan las relaciones unidireccionales, aunque la implementación de las mismas en ATL en la herramienta M2DAT se ha realizado de forma bidireccional. Las mismas reglas que se muestran a continuación, se pueden aplicar también en las relaciones bidireccionales.

En la parte izquierda de la Fig. 8 se muestra la regla de transformación de grafos para el caso de las asociaciones 1:N. Estas se identifican por el valor del atributo *upper,* incluido en la propiedad UML de la clase origen (②). La relación (unidireccional) se transforma incluyendo un atributo en el tipo estructurado correspondiente a la clase origen de la asociación (①'−②'). Este atributo será una colección de referencias al tipo estructurado definido a partir de la clase destino de la relación (②'−②'), es decir, la *Nested Table* contiene elementos de tipo *Ref*.

A la derecha de la figura se muestra la correspondiente implementación en ATL. El punto de entrada a esta regla asegura que solamente las asociaciones 1:N de UML sean transformadas aplicando esta regla (lin. 3-5). En la transformación se crea un atributo OR y una *Nested Table* (lin. 6-16). El tipo del atributo será la *Nested Table* creada a través de la invocación a la regla *lazy generateNestedTable* (lin. 17-25). De la misma manera se crea el tipo de los elementos de la colección (*Nested Table*) como tipos *Ref* a través de la invocación de la regla *lazy generateReferences* (lin. 26-33). Las dos reglas *lazy* que se utilizan son *unique*. Esto permite

asegurar que ningún tipo *Ref* ni ninguna *Nested Table* se encuentren duplicados. Si alguna de las reglas se invoca con los mismos parámetros, para crear nuevos tipos, las reglas devuelven la referencia al tipo creado la primera vez que se la invocó.

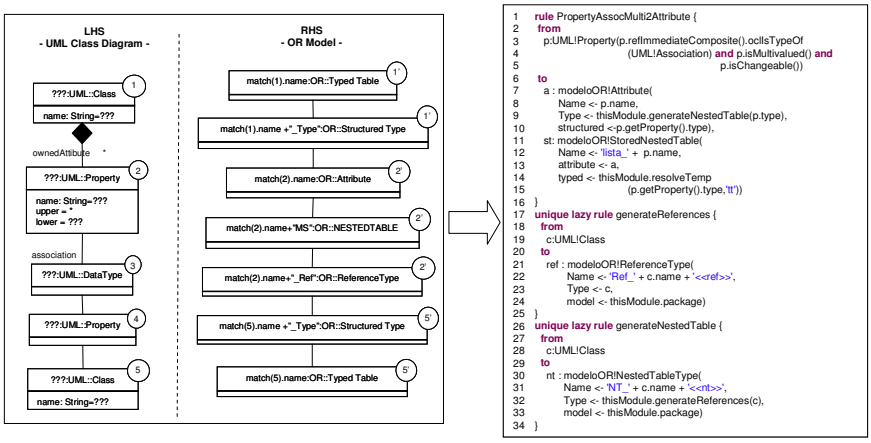

Fig. 8. Regla de Transformación de Asociaciones

La Fig. 9 muestra el resultado de transformar la asociación 1:N entre *Plan* y *Figure.* Además de los tipos estructurados y las tablas tipadas que se crean a partir de las clases *Plan* y *Figure*, se crea también el tipo *Ref* (*Ref_Figure*) y el tipo *Nested Table* (*NT_Figure*), que contiene una colección de elemento de tipo *Ref*. En el atributo de tipo colección *Have* de la clase *Plan* hace referencia al tipo *NT_Figure*.

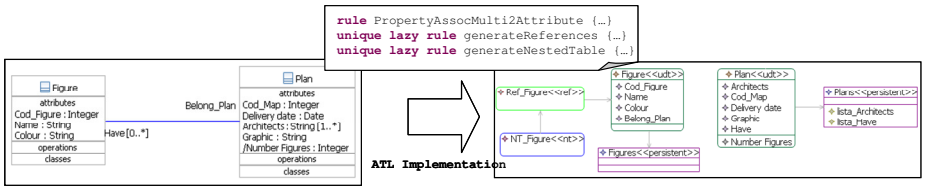

Fig. 9. Transformación de la Asociación 1:N *Have*

3.3 Modelo OR

Por último, en la Fig. 10 se muestra el PSM completo de la BDOR para este caso de estudio representado mediante el editor gráfico de la herramienta M2DAT. La figura se corresponde con el resultado de aplicar las reglas de transformación de ATL al PIM (Fig. 3) en M2DAT.

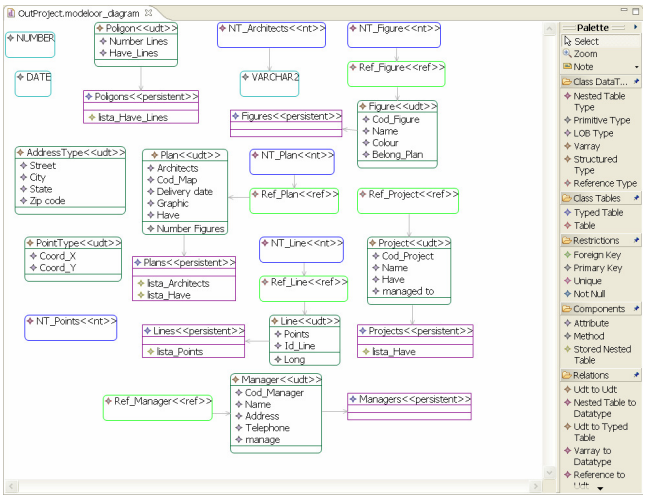

Fig. 10. PSM de la BDOR del caso de estudio

3.4 Generación de Código

El último paso del proceso de desarrollo es la generación de código. En realidad este paso es una nueva transformación, pero ésta de modelo a texto (M2T). Para esta tarea se ha utilizado el lenguaje MOFScript [16], un prototipo que a día de hoy es el que cuenta con más posibilidades de ser aceptado por la OMG como estándar para transformaciones M2T. Al ser la primera implementación disponible, es probablemente la más consolidada y de uso más extendido. Además su aprendizaje no requiere mucho esfuerzo: una vez que se han desarrollado transformaciones M2M, el paso a la generación de código con MOFScript resulta sencillo e intuitivo. Se puede encontrar más información sobre cómo configurar y ejecutar MOFScript en [16].

A continuación se mostrarán algunas partes del *script* que genera a partir del PSM el código SQL que implementa el esquema de BDOR en el producto Oracle10g. Se puede ver este *script* o transformación como un *parser* o analizador de modelos, que recorre el modelo del esquema de la BDOR (PSM) mientras va generando la salida (código SQL).

El *script* se codifica en una función principal o *main* que contiene un conjunto de reglas. Para cada tipo de elemento o clase del metamodelo de entrada (también llamados *context types* en MOFScript) se codifica una regla que permite serializar los objetos de dicho tipo.

```
88   // Auxiliary Function for Structured Type code generation
89  ecc StructuredType::generateStructured(  {
90      var texto String=""
91      if  self supertype Name size()=0
92          texto="CREATE OR REPLACE TYPE " + self Name replace "<<udt>>", "") + " AS OBJECT \r\n('
93      else
94          texto="CREATE OR REPLACE TYPE " + self Name replace "<<udt>>", "") + " UNDER " +
95          self supertype Name replace "<<udt>>", "") + "\r\n('
96          println texto
97
98       // Attributes
99       self attribute->forEach a ecc Attribute  {
...
128       // Typed Tables
129       self typed->forEach t ecc TypedTable {
130         println "CREATE TABLE " + t Name replace "<<persistent>>","") +
                " OF " + self Name replace "<<udt>>","") + "\r\n('
131         t generateTypedTable(
132          println "'
133       }
134    }
```

Las reglas simples se definen dentro del *main* mientras que las reglas complejas se definen como funciones auxiliares que son invocadas desde el *main*. Por ejemplo, la regla para la creación de tipos estructurados es probablemente una de las reglas más complejas, ya que encapsula una gran cantidad de semántica. Esta se codifica dentro de una función auxiliar *generateStructured.*

El código mostrado implementa esta función (por cuestiones de espacio se ha ocultado el código correspondiente a la transformación de atributos, lin.100-127). En primer lugar se inicializa la variable auxiliar que almacenará el código SQL (lin. 90). Luego se añade el código SQL para la creación de los tipos estructurados, separando aquellos que heredan de otros tipos (lin. 91-96). Luego se transforman los atributos (lin. 100-127), esta regla se ejecuta por cada atributo que pertenezca al tipo estructurado (lin. 99). Por último, se recorre la propiedad *typed* de los tipos estructurados para identificar las tablas tipadas a las que hacen referencia los tipos estructurados (lin. 129). De esta manera se comienza la creación de las tablas tipadas a través de la invocación a la regla *generateTypedTable* (lin. 130-131).

En el código que se muestra a continuación se ve la regla de transformación para los objetos *Nested Table*. Este código se encuentra dentro de la función *main*. La estructura *forEach* permite iterar sobre cada uno de los tipos *NestedTableType* del modelo origen (lin. 55). Como las *NestedTableType* dependen de las clases *DataType*, se recorren todos los *DataType* que las contengan. Luego se continua con los tipos primitivos (lin. 59-63), donde se da un tratamiento

especial a los tipos primitivos creados por el usuario (lin. 64-73). Por último, se escribe el código resultante en el archivo de salida (lin. 74).

```
54  //Nested Table code generation
55   self.datatype->forEach(n|eco!NestedTableType) {
56     var anterior:String;
57     var posterior:String;
58     var tipo:String;
59     if (n.Type.oclGetType()="PrimitiveType" or n.Type.oclGetType() ="LOBType");
60        if (n.Type.Size!=0)
61          tipo=n.Type.Name + "(" + n.Type.Size + ")"
62        else
63          tipo=n.Type.Name
64     else {
65          posterior=n.Type.Name.substringAfter(">>")
66          anterior=n.Type.Name.substringBefore("<<")
67          if (anterior=="" and posterior="");
68             tipo=n.Type.Name
69          else if (anterior="")
70                tipo=posterior
71             else
72                tipo=anterior
73       }
74     println("CREATE OR REPLACE TYPE " + n.Name.replace("<<nt>>","") + " AS TABLE OF " + tipo + ";")
75   }
```

En la Fig. 11 se muestra una pequeña parte del código SQL generado para el caso de estudio. En la parte superior de la figura se muestra una parte del modelo OR para el caso de estudio representado con el editor gráfico de la herramienta: el tipo estructurado *Poligon* y su correspondiente tabla tipada *Poligons*, además de los tipos *Line* y *NT_Line*. En la parte inferior de la figura se muestra el correspondiente código SQL generado a partir de la ejecución de las reglas de transformación (M2T) mostradas anteriormente.

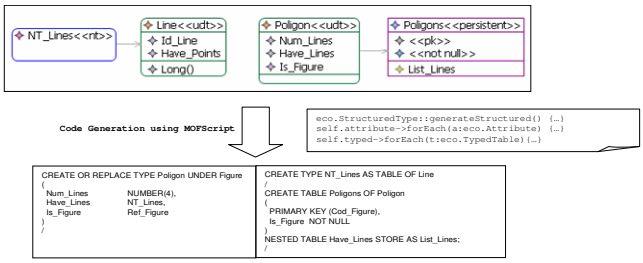

Fig. 11. Ejemplo de Generación de Código

4 Conclusiones y Trabajos Futuros

En este trabajo se ha completado la propuesta del proceso de desarrollo dirigido por modelos para BDOR dentro del marco de MIDAS. Para ello, se han implementado con ATL las reglas de transformación M2M previamente formalizadas, permitiendo la generación de un modelo OR a partir del modelo conceptual de datos, y las transformaciones M2T con MOFScript, generando el código SQL para el esquema de la BDOR. Como parte de la propuesta se ha definido también un DSL basado en MOF para el modelado de BDOR y un editor gráfico que implementa ese DSL.

El proceso de desarrollo propuesto se ha implementado como un módulo de M2DAT (*MIDAS MDA Tool*), la herramienta que integra todas las técnicas para la generación (semi)- automática de sistemas propuestas en MIDAS. Este trabajo ha servido como prueba de implementación, ya que a partir del mismo se han aprendido lecciones y adquirido buenas prácticas con respecto a la implementación. Cuando el resto de los módulos se encuentren implementados y validados, se procederá a la integración de los mismos en la herramienta M2DAT.

Actualmente se está trabajando en la utilización de modelos *weaving* para realizar anotaciones al modelo OR. De esta manera se podría parametrizar el modelo de transformación, con lo que el proceso de desarrollo de BDOR se podría personalizar para cada caso específico. Así, el usuario

podrá elegir, por ejemplo ante un atributo multivaluado, su transformación al tipo colección de Oracle que el diseñador elija, es decir, o bien a un tipo *Nested Table* o un tipo *VArray*.

Agradecimientos

Este trabajo se ha realizado en el marco del proyecto GOLD, financiado por el Ministerio Español de Educación y Ciencia (TIN2005-00010/) y el proyecto M-DOS (URJC-CM-2007-CET-1607) co-financiado por la Universidad Rey Juan Carlos y la Comunidad de Madrid (España).

Referencias

1. Acerbis, R., Bongio, A., Brambilla, M. y Butti, S., *WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications*, en Web Engineering, 2007, 501-505.
2. Baresi, L. y Heckel, R. 2002. *Tutorial Introduction to Graph Transformation: A Software Engineering Perspective.* En Proceedings of the First international Conference on Graph Transformation). LNCS 2505. Springer-Verlag, London, pp. 402-429, 2002.
3. Bertino, E. y Marcos, E. *Object Oriented Database Systems. In: Advanced Databases: Technology and Design*, O. Díaz y M. Piattini (Eds.). Artech House, 2000.
4. Bézivin, J. *Some Lessons Learnt in the Building of a Model Engineering Platform*. 4th Workshop in Software Model Engineering (WISME), Montego Bay, Jamaica (2005)
5. Bollati, V.A, Vara, J.M., Vela, B. y Marcos, E. *Una Aproximación Dirigida por Modelos para el Desarrollo de Esquemas XML*. XIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2008). Aceptado para su publicación.
6. Chen, P.P. *The Entity-Relationship Model – Toward a Unified View of Data*. ACM Transactions on Database Systems, Vol. 1, No. 1. Marzo 1976, pp. 9-36, 1976.
7. Didonet Del Fabro, M.: *Metadata management using model weaving and model transformation*. Ph.D. Tesis Universidad de Nantes (2007).
8. IBM DB2 Universal Database. http://www-306.ibm.com/software/data/db2/.
9. ISO / IEC 9075 *Standard, Information Technology – Database Languages – SQL:2003*, International Organization for Standardization, 2003.
10. Jouault, F. y Kurtev, I. *Transforming Models with ATL*. En: Proc. of the Model Transformations in Practice Workshop. MoDELS Conference, Jamaica. 2005.
11. Marcos, E. Vela, B., Cáceres, P. y Cavero, J.M. *MIDAS/DB: a Methodological Framework for Web Database Design*. DASWIS 2001. Yokohama (Japón), Noviembre, 2001. LNCS 2465, Springer-Verlag, pp. 227-238, Septiembre, 2002.
12. Marcos, E., Vela, B. y Cavero, J.M. *Extending UML for Object-Relational Database Design*. Fourth Int. Conference on the Unified Modeling Language, UML 2001. Toronto (Canada), LNCS 2185, Springer-Verlag, pp. 225-239. Octubre, 2001.
13. Marjan, M., Jan, H., Anthony, M.S.: *When and how to develop domain-specific languages*. ACM Comput. Surv. 37 (2005) pp. 316-344.
14. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Final Adopted Specification 07/07/07. Recuperado de: http://www.omg.org.
15. Microsoft SQL Server. Recuperado de: http://www.microsoft.org/sql/.
16. Oldevik, J., Neple, T., Grønmo, R., Aagedal, J. y Berre, A.-J. *Toward Standardised Model to Text Transformations. Model-driven Architecture – Foundations and Applications*, pp. 239-253, 2005.
17. OMG. Object Constraint Language, OMG Available Specification Versión 2.0. Formal: 01/05/2006. Recuperado de www.omg.org/docs/ptc/06-05-01.pdf
18. OMG. *MDA Guide Version 1.0.* Document number omg/2003-05-01. Ed.: Miller, J. y Mukerji, J. Recuperado de: http://www.omg.com/mda, 2003.
19. Oracle Corporation. *Oracle Database 10g. Release 2 (10.2)*. Recuperado de: www.oracle.com.
20. Stahl, T., Volter, M. y Czarnecki, K. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.
21. Stonebraker, M. y Brown, P. *Object-Relational DBMSs*. Tracking the Next Great Wave. Morgan Kauffman, 1999.
22. *Unified Modeling Language* 2 (UML2). Recuperado de: http://www.eclipse.org/modeling/mdt/?project=uml2
23. Vara, J.M., Vela, B., Cavero, J.M. y Marcos, E. *Model transformation for object-relational database development*. SAC '07: Proceedings of the 2007 ACM symposium on Applied computing, pp.1012-1019, 2007.
24. Vara, J.M., De Castro, V., Caceres, P., Marcos, E., Arquitectura de MIDAS-CASE : una herramienta para el desarrollo de SIW basada en MDA. IV Jornadas Iberoamericanas en Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'04). pp.83-92
25. Vara, J.M., De Castro, V. y Marcos, E. WSDL automatic generation from UML models in a MDA framework In International Journal of Web Services Practices. Volume 1 – Issue 1 & 2. Noviembre 2005, pp.1 – 12. 2005
26. Vela B., Acuña C. y Marcos E. A *Model Driven Approach for XML Database Development*, 23rd. International Conference on Conceptual Modelling (ER2004). LNCS 3288. Springer Verlag, pp. 780-794. 2004.
27. Vela B. y Marcos E. *Extending UML to represent XML Schemas.* The 15th Conference On Advanced Information Systems Engineering. CAISE'03 FORUM. Klagenfurt/Velden (Austria). 16-20 Junio 2003. Ed: J. Eder, T. Welzer. Short Paper Proceedings, 2003.